***Machine Learning***

***answer of assignments 1***

***Dr. Farahani***

***Hasan Roknabady – 99222042***

## *Descriptive part*

# Ans 1 :

Yes, gradient descent can get stuck in a local minimum when training a logistic regression model.

Logistic regression is a type of supervised learning algorithm used for binary classification problems. The aim of logistic regression is to find the best set of weights (also called parameters) that will give the lowest prediction error. Gradient descent is a popular optimization algorithm used to find the optimal weights for logistic regression.

Gradient descent works by iteratively updating the weights in the direction of the negative gradient of the cost function. The cost function is a measure of the difference between the predicted and actual values of the target variable. The aim of gradient descent is to minimize this cost function.

However, the cost function for logistic regression is non-convex, which means that it can have multiple local minima. A local minimum is a point where the cost function is at its lowest, but it is not the global minimum. Gradient descent can get stuck in a local minimum because it only updates the weights in the direction of the steepest descent, which may not necessarily lead to the global minimum.

When gradient descent gets stuck in a local minimum, it means that the weights have converged to a sub-optimal set of values, and further iterations of gradient descent will not improve the performance of the logistic regression model.

To prevent gradient descent from getting stuck in a local minimum, several techniques can be used, such as random initialization of the weights, early stopping, and using a different optimization algorithm, such as stochastic gradient descent.

# Ans 2:

If we are using polynomial regression and you notice a large gap between the training error and the validation error, it is an indication that your model is overfitting the training data. Overfitting occurs when a model learns the noise and idiosyncrasies of the training data and does not generalize well to new data.

In the case of polynomial regression, the model is trying to fit a polynomial function to the training data, and if the degree of the polynomial is too high, the model can overfit the training data, leading to a large gap between the training error and the validation error.

Here are three ways to solve this problem:

1. Regularization: Regularization is a technique that penalizes large coefficients in the model to prevent overfitting. In polynomial regression, Lasso or Ridge regression can be used for regularization. Lasso regression adds an L1 penalty term to the cost function, which encourages the model to have fewer non-zero coefficients, whereas Ridge regression adds an L2 penalty term to the cost function, which discourages the model from having large coefficients. By adding a regularization term to the cost function, the model will be forced to simplify its learned function and avoid overfitting.
2. Reduce the model complexity: Another way to reduce overfitting in polynomial regression is to reduce the degree of the polynomial. This will make the model less complex and more generalizable. A lower-degree polynomial may still capture the underlying trend in the data without overfitting to the noise.

3. Increase the size of the training data: Increasing the size of the training data can help in reducing overfitting. By providing more training data, the model can learn more generalized patterns that can be applied to new data. When the size of the training data is increased, the model is less likely to overfit the training data, and the gap between the training error and the validation error will be reduced.

In summary, a large gap between the training error and the validation error in polynomial regression indicates overfitting. To solve this, you can use regularization, reduce the model complexity, or increase the size of the training data.

## Ans 3 :

using Ridge regression and notice that both the training error and the validation error are almost equal and fairly high, it is an indication that the model suffers from high bias. High bias occurs when a model is too simple and unable to capture the complexity of the underlying data, resulting in underfitting.

In the case of Ridge regression, the regularization hyperparameter α controls the trade-off between the size of the coefficients and the magnitude of the error. A large α value results in smaller coefficients, which in turn results in a simpler model, whereas a small α value results in larger coefficients and a more complex model. Therefore, in the case of high bias, we need to reduce the regularization hyperparameter α to allow the model to become more complex and fit the data better.

Reducing the value of α will decrease the amount of regularization applied to the model, allowing the model to fit the data better and reducing the bias. However, it is important to be careful not to reduce α too much, as this may result in overfitting, where the model fits the noise and idiosyncrasies of the training data and does not generalize well to new data. Therefore, it is recommended to start with a small α value and gradually increase it until the validation error starts to increase again.

In summary, if you notice that the training error and the validation error are almost equal and fairly high when using Ridge regression, it indicates high bias, and you should reduce the regularization hyperparameter α to allow the model to become more complex and fit the data better. However, it is important to avoid reducing α too much to prevent overfitting.

# Ans 4 :

- ## First Segment: Ridge regression instead of plain linear regression (i.e., without any regularization)?

Ridge regression is a regularized form of linear regression that adds a penalty term to the cost function in order to prevent overfitting. In contrast, plain linear regression does not include any regularization terms and can be prone to overfitting, especially when the number of features in the dataset is large or when the features are highly correlated with each other. Here are some specific reasons why you might want to use Ridge regression instead of plain linear regression:

1) Handling multicollinearity: When the features in the dataset are highly correlated with each other, the coefficient estimates from plain linear regression can have high variance, which can lead to overfitting. Ridge regression adds a penalty term to the cost function that shrinks the magnitude of the coefficient estimates, reducing their variance and improving their stability.

2) Improved generalization performance: Ridge regression can improve the generalization performance of the model by reducing the complexity of the model, which can help to prevent overfitting. By adding a penalty term to the cost function, Ridge regression encourages the model to find a set of coefficients that are small and smooth, reducing the chance of overfitting.

3) Better accuracy: Ridge regression can lead to better accuracy in prediction than plain linear regression, especially when the dataset has a large number of features. By reducing the complexity of the model and preventing overfitting, Ridge regression can help to improve the accuracy of the model on both the training and validation sets.

Overall, Ridge regression is a useful regularization technique that can improve the performance of a linear regression model, especially when the dataset has many features or when the features are highly correlated with each other. By adding a penalty term to the cost function, Ridge regression can help to prevent overfitting and improve the generalization performance of the model, leading to better accuracy and more robust predictions.

## Second Segment : Lasso instead of ridge regression?

Lasso (short for "least absolute shrinkage and selection operator") is a regularization technique that can be used in place of Ridge regression. While both techniques add a penalty term to the cost function in order to prevent overfitting, the penalty term used in Lasso is different from the one used in Ridge regression. Specifically, Lasso uses the L1 norm of the coefficient vector as the penalty term, whereas Ridge regression uses the L2 norm.

Here are some specific reasons why you might want to use Lasso instead of Ridge regression:

1. Feature selection: Lasso can be used for feature selection, as it tends to shrink the coefficients of less important features to zero, effectively removing them from the model. This can be useful when dealing with datasets with a large number of features, as it can help to reduce the dimensionality of the problem and improve the interpretability of the model.

2. Simplicity: Lasso tends to produce sparse models with fewer nonzero coefficients than Ridge regression. This can make the model simpler and more interpretable, as it can help to identify the most important features that contribute to the prediction.
3. Better performance on sparse datasets: Lasso performs better than Ridge regression when dealing with sparse datasets, where most of the coefficients are zero. In this case, Lasso can effectively eliminate the irrelevant features, leading to a better prediction performance.

Overall, Lasso can be a useful regularization technique when dealing with high-dimensional datasets or when feature selection is desired. By using the L1 norm as the penalty term, Lasso can encourage sparsity and improve the interpretability of the model, while still preventing overfitting and improving the generalization performance of the model. However, it is worth noting that Lasso can be sensitive to correlated features, and may not perform well in cases where there are many highly correlated features.

Additionally, here are some other reasons why you might want to use Lasso instead of Ridge regression:

4. Better interpretation of coefficients: As Lasso shrinks the coefficients towards zero, it is easier to interpret the magnitude and direction of the remaining non-zero coefficients. This can be particularly useful in settings where you are trying to understand the underlying relationships between the predictors and the response variable.
5. Better performance in situations with few predictors: When you have a relatively small number of predictors, Lasso can be more effective than Ridge regression at identifying which predictors are important for predicting the response variable.
6. Reducing multicollinearity: In situations where you have highly correlated predictors, Lasso can be more effective than Ridge regression at selecting a subset of predictors that are not highly correlated with one another. This can help to reduce multicollinearity in the model, which can improve its overall performance.

In summary, while both Ridge regression and Lasso are useful regularization techniques that can help to prevent overfitting and improve the generalization performance of a model, Lasso has some specific advantages over Ridge regression in terms of feature selection, simplicity, and performance in sparse datasets. However, the choice of regularization technique ultimately depends on the specific problem you are trying to solve and the characteristics of your data.

## • Third Segment: Elastic net instead of lasso regression?

Elastic Net is a regularization technique that combines the L1 and L2 penalties into a single objective function, and allows for tuning of the relative contribution of each penalty. It provides a balance between the strengths of Lasso and Ridge regression, and can be particularly useful in situations where there are many correlated predictors. Here are some specific reasons why you might want to use Elastic Net instead of Lasso regression:

1. Robustness to correlated predictors: Elastic Net can handle situations where there are many correlated predictors, whereas Lasso may not perform well in such cases. By combining the L1 and L2 penalties, Elastic Net can effectively shrink the coefficients of correlated predictors while still preserving some information about them.

2. Feature selection and model simplicity: Like Lasso, Elastic Net can perform feature selection by shrinking some coefficients to zero. However, it can also provide a balance between feature selection and model complexity by allowing for non-zero coefficients for some correlated predictors.
3. Better prediction performance: Elastic Net can often outperform Lasso in terms of prediction accuracy, particularly when dealing with high-dimensional datasets with many correlated predictors. This is because Lasso can be too aggressive in shrinking coefficients to zero, leading to loss of information.
4. Flexibility in parameter tuning: Elastic Net allows for tuning of the relative contributions of the L1 and L2 penalties, which provides more flexibility in choosing the appropriate regularization strength for a given problem.

In summary, Elastic Net is a useful regularization technique that provides a balance between the strengths of Lasso and Ridge regression. It can handle correlated predictors better than Lasso, perform feature selection and model simplicity, and provide better prediction performance than Lasso in high-dimensional datasets. However, the choice of regularization technique ultimately depends on the specific problem you are trying to solve and the characteristics of your data.

## Ans 7 : Compare bootstrapping with cross-validation. In which conditions we should use bootstrapping?

Bootstrapping and cross-validation are two popular techniques for estimating the performance of a machine learning model on new data. Although they have some similarities, they differ in several important ways.

Bootstrapping involves repeatedly resampling the original dataset with replacement to create new datasets of the same size as the original dataset. Each resampled dataset is used to fit a new model, and the results are combined to estimate the overall performance of the model. The idea is that by resampling the data, we can simulate the process of drawing new samples from the same population and get a better estimate of the model's performance on new data.

Cross-validation, on the other hand, involves dividing the original dataset into k equally sized subsets, or folds. One fold is held out as the validation set, while the remaining k-1 folds are used to fit the model. This process is repeated k times, with each fold used as the validation set once. The results are averaged to estimate the overall performance of the model.

Here are some differences between bootstrapping and cross-validation:

1. Sample size: Bootstrapping can be used when the sample size is too small for cross-validation to be effective. In general, cross-validation requires a larger sample size to provide reliable estimates of the model's performance.
2. Computational complexity: Bootstrapping can be more computationally intensive than cross-validation, especially if a large number of resamples are used. Cross-validation is often faster and easier to implement.

3. Bias and variance: Bootstrapping tends to have lower bias than cross-validation, but higher variance. This means that bootstrapping can be more accurate in estimating the performance of a model, but the estimates can be more variable from one resample to the next.
4. Use case: Bootstrapping is often used to estimate confidence intervals or uncertainty intervals for model parameters, while cross-validation is used to estimate the performance of a model on new data.

In general, bootstrapping is more appropriate when the sample size is small, when estimating confidence intervals is important, or when the goal is to estimate the variability of the model parameters. Cross-validation is more appropriate when the sample size is large enough, when estimating the performance of the model on new data is the main objective, or when selecting the best model among several alternatives.

In summary, bootstrapping and cross-validation are both valuable techniques for estimating the performance of a machine learning model on new data. The choice between them depends on the specific problem at hand, the size of the dataset, and the specific objectives of the analysis.

## Ans 8: Explain nested cross-validation and 5x2 cross-validation in detail and when we should use them.

Nested cross-validation and 5x2 cross-validation are two commonly used techniques for evaluating machine learning models. They are especially useful when the dataset is small or when hyperparameter tuning is required.

Nested cross-validation involves using an outer loop and an inner loop of cross-validation. The outer loop is used to split the data into training and testing sets, while the inner loop is used to tune the hyperparameters of the model. The idea is to prevent overfitting by using a different set of hyperparameters for each fold of the outer loop.

Here is the general process of nested cross-validation:

1. Split the data into k folds for the outer loop.
2. For each fold in the outer loop: a. Split the data into k-1 folds for the inner loop. b. For each combination of hyperparameters, train the model on the k-1 folds and evaluate its performance on the remaining fold. c. Select the hyperparameters that give the best performance on the k-1 folds. d. Train the model on all of the data except for the fold in the outer loop, using the selected hyperparameters. e. Evaluate the performance of the model on the fold in the outer loop.
3. Compute the average performance of the model over all folds in the outer loop.

Nested cross-validation is useful when the dataset is small or when the model has many hyperparameters that need to be tuned. It provides a more accurate estimate of the model's performance on new data than regular cross-validation, as it uses a different set of hyperparameters for each fold of the outer loop.

5x2 cross-validation is a variant of nested cross-validation that involves splitting the data into two parts, each with 5 folds. The idea is to repeat the process twice, with each part used as the training set and the other part used as the testing set. The performance of the model is averaged over the two repetitions.

Here is the general process of 5x2 cross-validation:

1. Split the data into two parts, each with 5 folds.
2. For each part: a. For each fold: i. Train the model on the remaining 4 folds. ii. Evaluate the performance of the model on the remaining fold. b. Compute the average performance of the model over all 5 folds.
3. Compute the average performance of the model over the two parts.

5x2 cross-validation is useful when the dataset is small and the model has many hyperparameters that need to be tuned. It provides a more reliable estimate of the model's performance than regular cross-validation, as it repeats the process with different training and testing sets.

In summary, nested cross-validation and 5x2 cross-validation are useful techniques for evaluating machine learning models, especially when the dataset is small or when hyperparameter tuning is required. Nested cross-validation is more appropriate when the model has many hyperparameters that need to be tuned, while 5x2 cross-validation is more appropriate when a reliable estimate of the model's performance is required.

Both techniques aim to prevent overfitting and provide a more accurate estimate of the model's performance on new data. Nested cross-validation involves using an outer loop and an inner loop of cross-validation to tune the hyperparameters, while 5x2 cross-validation involves splitting the data into two parts and repeating the process twice with different training and testing sets.

Nested cross-validation can be computationally expensive, especially when the model has many hyperparameters that need to be tuned. However, it provides a more accurate estimate of the model's performance on new data than regular cross-validation, as it uses a different set of hyperparameters for each fold of the outer loop.

5x2 cross-validation is less computationally expensive than nested cross-validation, as it involves repeating the process with different training and testing sets. It provides a more reliable estimate of the model's performance than regular cross-validation, as it uses two different sets of training and testing data.

In general, if the dataset is small or if hyperparameter tuning is required, it is recommended to use nested cross-validation or 5x2 cross-validation instead of regular cross-validation. Nested cross-validation is more appropriate when the model has many hyperparameters that need to be tuned, while 5x2 cross-validation is more appropriate when a reliable estimate of the model's performance is required.

To give a more detailed explanation, let's consider each technique separately:

1. Nested Cross-Validation

Nested cross-validation is a technique that is used to tune the hyperparameters of a model and provide a more accurate estimate of its performance on new data. It involves an outer loop and an inner loop of cross-validation. The outer loop is used to split the data into training and testing sets, while the inner loop is used to tune the hyperparameters of the model.

The outer loop involves splitting the data into k-folds, where k is typically set to 5 or 10. Each fold is used once as the testing set, while the remaining k-1 folds are used as the training set. The model is trained on the training set and evaluated on the testing set. This process is repeated k times, with each fold used once as the testing set.

The inner loop is used to tune the hyperparameters of the model. It involves splitting the training set from the outer loop into l-folds, where l is typically set to 5 or 10. Each fold is used once as the validation set, while the remaining l-1 folds are used as the training set. The model is trained on the training set and evaluated on the validation set. This process is repeated l times, with each fold used once as the validation set.

The best set of hyperparameters is selected based on the average performance on the validation sets from the inner loop. Once the best set of hyperparameters is selected, the model is trained on the entire training set from the outer loop using these hyperparameters. The performance of the model is then evaluated on the testing set from the outer loop.

Nested cross-validation provides a more accurate estimate of the model's performance on new data than regular cross-validation, as it uses a different set of hyperparameters for each fold of the outer loop. However, it can be computationally expensive, especially when the model has many hyperparameters that need to be tuned.

2. 5x2 Cross-Validation

5x2 cross-validation is a technique that is used to provide a more reliable estimate of the model's performance than regular cross-validation. It involves splitting the data into two parts and repeating the process twice with different training and testing sets.

The first step involves randomly splitting the data into two parts, typically with a 50/50 split. Each part is then split into five equal-sized folds, and the model is trained and tested using each fold as the testing set and the remaining folds as the training set. This process is repeated twice, with different training and testing sets.

The performance of the model is then evaluated based on the two sets of results obtained from the two repetitions. This provides a more reliable estimate of the model's performance than regular cross-validation, as it uses two different sets of training and testing data.

5x2 cross-validation is less computationally expensive than nested cross-validation, as it involves repeating the process with different training and testing sets. However, it may not be as accurate as nested cross-validation, especially when the model has many hyperparameters that need to be tuned.

In general, if a reliable estimate of the model's performance is required, it is recommended to use 5x2 cross-validation. If hyperparameter tuning is required, it is recommended to use nested cross-validation.

## Ans 9 : How can we compare different models using statistical significance tests?

To compare different models using statistical significance tests, we can perform hypothesis testing. The most common way to perform hypothesis testing is to use a null hypothesis and an alternative hypothesis. The null hypothesis is the assumption that there is no significant difference between the models being compared. The alternative hypothesis is the opposite, that there is a significant difference between the models.

One common statistical significance test for comparing two models is the paired t-test. This test involves comparing the mean difference between the performance of the two models on a validation dataset against a null hypothesis that the mean difference is zero. The t-test calculates a p-value, which indicates the probability of observing the data assuming the null hypothesis is true. If the p-value is less than a predetermined significance level (usually 0.05), we reject the null hypothesis and conclude that there is a significant difference between the two models.

Another commonly used test is the McNemar test, which is used to compare the performance of two models on a binary classification task. The McNemar test involves comparing the number of instances in which one model outperforms the other against a null hypothesis that the difference in performance is due to chance. If the p-value is less than the predetermined significance level, we reject the null hypothesis and conclude that there is a significant difference between the models.

In addition to these tests, we can also use cross-validation to compare models. One common approach is to use repeated k-fold cross-validation, where we repeatedly split the data into k-folds and train each model on k-1 folds and test on the remaining fold. We can then compute the mean and standard deviation of the performance metrics for each model and compare them using statistical tests such as the paired t-test or ANOVA.

Overall, statistical significance tests provide a rigorous way to compare different models and determine if the difference in performance is due to chance or a real difference in the models. However, it is important to keep in mind that statistical tests can be influenced by factors such as the size of the dataset, the choice of performance metric, and the distribution of the data.

Let's consider a practical example of comparing two models for a binary classification task. Suppose we have a dataset of customer reviews for a product and we want to predict whether a review is positive or negative. We have two models, a logistic regression model and a decision tree model, and we want to compare their performance.

To compare the models using statistical significance tests, we can use cross-validation. Let's say we decide to use 10-fold cross-validation, where we split the data into 10 folds and train each model on 9 folds and test on the remaining fold. We can then repeat this process 10 times, each time using a different fold as the validation set, and compute the mean and standard deviation of the performance metrics for each model.

Suppose we use accuracy as the performance metric. After running the cross-validation, we find that the logistic regression model has a mean accuracy of 0.85 with a standard deviation of 0.02, while the decision tree model has a mean accuracy of 0.81 with a standard deviation of 0.03. From these results, we can see that the logistic regression model performs better on average than the decision tree model.

To determine if the difference in performance is significant, we can use a statistical significance test such as the paired t-test. The paired t-test involves comparing the mean difference between the performance of the two models on each fold against a null hypothesis that the mean difference is zero. Suppose we find that the p-value of the t-test is 0.02, which is less than the significance level of 0.05. This means that we can reject the null hypothesis and conclude that there is a significant difference between the two models.

Overall, this example illustrates how we can use cross-validation and statistical significance tests to compare the performance of different models and determine if the difference is significant.

## Ans 11 : Suppose the features in your training set have very different scales. Which algorithms *(Gradient Descent, Normal Equation, SVD)* might suffer from this, and how? What can you do about it?

the features in a training set have different scales, that can cause issues for some machine learning algorithms that use numerical optimization techniques such as Gradient Descent, Normal Equation, and SVD. This is because these algorithms rely on taking steps towards the minimum of the cost function, which can be greatly affected by the scale of the features.

Gradient Descent:

Gradient Descent is a numerical optimization algorithm that aims to minimize the cost function by iteratively adjusting the model parameters. If the features have different scales, the gradient descent algorithm may converge very slowly or not at all, making the optimization process ineffective.

Specifically, if one feature has a much larger scale than the others, then its influence on the cost function will dominate, and the algorithm will focus on minimizing this feature's contribution, potentially ignoring the other

features altogether. As a result, the learning process will be slow or may never converge to the global minimum.

Normal Equation:

Normal equation is a closed-form solution that directly computes the optimal model parameters without the need for iterative optimization. However, when features have very different scales, it can cause the condition number of the feature matrix to be very large, which may lead to numerical instability or inaccuracy in the solution.

SVD:

SVD (Singular Value Decomposition) is a matrix factorization technique that is used for dimensionality reduction and feature extraction. SVD can also be used in regression problems to obtain a closed-form solution. However, similar to Normal Equation, SVD can also suffer from numerical instability when the feature matrix has a large condition number due to different feature scales.

To address the problem of different feature scales, feature scaling techniques can be used. One common approach is to normalize the features by subtracting the mean and dividing by the standard deviation. This method, known as standardization, ensures that each feature has a mean of zero and a standard deviation of one. Another approach is to scale the features to a specific range, such as [0, 1] or [-1, 1]. This method is known as normalization.

In summary, it is important to consider the scale of the features in a training set when using numerical optimization techniques such as Gradient Descent, Normal Equation, and SVD. Feature scaling techniques can be used to address this issue and improve the effectiveness of these algorithms.

Let's consider a regression problem where we want to predict the price of a house based on its features such as the number of bedrooms, square footage, and the distance from the city center.

Suppose the number of bedrooms has a scale of 1-5, the square footage has a scale of 500-3000, and the distance from the city center has a scale of 1-50 miles. In this case, the features have very different scales, and it can cause issues for some machine learning algorithms.

If we apply the Gradient Descent algorithm without scaling the features, it may not converge to the global minimum, and the optimization process may become ineffective. This is because the features with a larger scale (square footage and distance from the city center) will have a more significant influence on the cost function than the number of bedrooms.

To address this issue, we can use feature scaling techniques such as normalization or standardization. For example, we can scale the number of bedrooms to a range of 0-1, the square footage to a range of 0-1, and the distance from the city center to a range of 0-1. This will ensure that all features have similar scales and will have an equal influence on the cost function.

By applying feature scaling, the Gradient Descent algorithm can converge more efficiently, and the prediction model will be more accurate. Therefore, it is crucial to consider feature scaling when dealing with machine learning problems that involve features with different scales.